## Remarks

Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 1, 2, 4-11, 13, 15-26, 28-36, 38, 40-42, and 44 are pending in the application. No claims have been allowed. Claims 1, 11, 23, 35, 41, and 44 are independent. Claims 1, 11, 13, 15, 16, 23, 28, 35, 38, 41, and 44 have been amended. Claims 3, 12, 14, 27, 37, 39, and 43 have been canceled without disclaimer and without prejudice to pursuing in a continuing application.

## Cited Art

The Action applies the following cited art: U.S. Pat. No. 6,330,717 to Raverdy et al. ("Raverdy").

## Rejections under 35 U.S.C. § 101

The Action rejects claims 3, and 35-43 under 35 U.S.C. § 101 as allegedly directed toward non-statutory subject matter.

Regarding claim 3, Applicants respectfully disagree with the § 101 rejection. However, in order to expedite prosecution, Applicants have canceled claim 3.

Regarding claims 35 and 41, the Examiner argues that the claims are directed to "non-statutory subject matter because the claims merely recite a data structure and/or program listing per se, thus they are non-functional descriptive materials." Action, page 3, citing the "Interim Guidelines" signed October 26, 2005, at ANNEX IV(b). Applicants respectfully disagree. As stated by the Interim Guidelines at ANNEX IV(a), "In contrast, *a claimed computer-readable medium encoded with a data structure* defines structural and functional interrelationships between the data structure and the computer software and hardware components which permit the data structure's functionality to be realized, *and is thus statutory*." (emphasis added). Because claims 35 and 41 recite a data structure or software code stored on a "computer readable storage medium," claims 35 and 41 are statutory according to the Interim Guidelines at ANNEX IV(a). Furthermore, because claims 35 and 41 are statutory, their respective dependent claims (36, 38, 40, and 42) are as well.

## Claim Rejections under 35 USC § 102

The Action rejects claims 1-44 under 35 USC 102(b) as being anticipated by Raverdy. Applicants respectfully submit the claims are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (*See* MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn.

Claim 1

Amended claim 1 reads as follows (emphasis added):

> receiving invocations of a plurality of software development scenario class extension sets comprising extensions for respective software development scenarios to be implemented by the extended version of the software, *wherein the respective software development scenarios specify a plurality of target processor architectures and a plurality of compilation scenarios*;

Claim 1 has been amended to recite, "wherein the respective software development scenarios specify a plurality of target processor architectures and a plurality of compilation scenarios." Support for the amendment to claim 1 can be found, for example, in the Application at page 6, line 16 to page 7, line 8, and page 21, lines 1-13.

Raverdy's description of an adaptive software platform for general purpose distributed applications does not teach or suggest "receiving invocations of a plurality of software development scenario class extension sets comprising extensions for respective software development scenarios to be implemented by the extended version of the software, *wherein the respective software development scenarios specify a plurality of target processor architectures and a plurality of compilation scenarios*" as recited by claim 1.

Raverdy describes a platform for distributed applications. The platform includes "reflective methods" that "allow a designer to customize the execution environment of an object at run-time" and "adaptive methods" that "allow a designer to provide several implementations for the same method and choose the most effective implementation at run-time." Raverdy, col. 2, lines 39-49. Raverdy also describes an "adaptation manger for configuring and managing the distributed application's execution environment." Raverdy, col. 2, lines 50-61. Raverdy describes a use of the platform as being able to adapt to various distributed environments, such as the Internet and low-bandwidth mobile networks. Raverdy, col. 5, lines 37-49. Raverdy

further describes how the compiler supports automatic extension using adaptive class definitions in order to maintain consistency of replicated instances in a distributed environment and to use different methods based on available bandwidth. Raverdy, col. 8, line 32 to col. 9, line 31.

As the above description of Raverdy clearly illustrates, Raverdy describes a platform for adapting distributed applications (e.g., network communication and replication). Raverdy does not teach or suggest "receiving invocations of a plurality of software development scenario class extension sets comprising extensions for respective software development scenarios to be implemented by the extended version of the software, *wherein the respective software development scenarios specify a plurality of target processor architectures and a plurality of compilation scenarios*" as recited by claim 1. In fact, Raverdy has no description of "target processor architectures" or "compilation scenarios."

For at least these reasons, Raverdy, does not teach or suggest all limitations of claim 1. Therefore, claim 1 should be allowable.

Claim 11

Claim 11 has been amended with language from dependent claims 12 and 14. Amended claim 11 reads as follows (emphasis added):

> receiving in an object description language definitions of extensions to classes of the core version of the software according to the configuration of the extended version of the software, wherein the classes of the core version of the software are indicated in the object description language as statically extensible prior to compile time or dynamically extensible at runtime, *wherein the classes of the core version of the software indicated as being statically extensible are processed together with their corresponding extensions, and wherein the classes of the core version of the software indicated as being dynamically extensible are processed separate from their corresponding extensions*;

For example, the Application at page 18, line 25 to page 20, line 9, and Fig. 10A, describes dynamically extensible class definitions as follows:

> Object description language for providing dynamically extensible class definitions
>
> One disadvantage of the static extensions is that it requires compiling the core framework and the extension together for generating one single file executable by a computer. This means those that are providing the extensions also need to have access to the source code of the core framework for recompiling it along with the extensions. To avoid this situation, which may not be desirable for a number of different reasons, extended class definitions may be generated at

runtime.

...

As noted above, in the case of dynamically generating an extended class the core class definitions 1010 and the class extension definitions 1020 are processed to create separate source code representations of the core class definitions and their extensions. Also, these separate source code representations are later compiled separately to generate separate files executable by a computer. In this case, unlike the static extensions described above, class extension members needed to extend a core class definition are not simply added to the source code header files with the extended class definitions. Instead, as shown in FIG. 10C, other code 1031 may be added to GET and SET class extensions that the dynamically extensible class definition expects to be added to the core class definition 1010 at runtime. Thus, in comparison to a static extension, the dynamic extensions may have the added overhead of having to execute some additional procedures 1031 in order to appropriately extend a class definition at runtime. Thus, typically the dynamic extensibility reduces the speed of a process but on the other hand provides for additional flexibility for providing extensions because in this approach the core and the extensions may be compiled separately. This allows for third parties to provide extensions to core class definitions without needing access to the source code of the core framework.

Raverdy's description of an adaptive software platform for general purpose distributed applications does not teach or suggest "*wherein the classes of the core version of the software indicated as being statically extensible are processed together with their corresponding extensions, and wherein the classes of the core version of the software indicated as being dynamically extensible are processed separate from their corresponding extensions*" as recited by claim 11.

Raverdy describes an "adaptation manger for configuring and managing the distributed application's execution environment." Raverdy, col. 2, lines 50-61. The "adaptation manager" controls adaptation policies, by adapting objects, while the application is running. Raverdy, col. 10, lines 21-53. For example, an adaptive video channel object can be adapted while the application is running (at run-time) to maintain performance. Raverdy, col. 10, lines 35-43. In order for the adaptation manager to adapt the applications, the objects are registered with the adaptation manager. Raverdy, col. 8, lines 22-24.

The "adaptation manager" described by Raverdy, which adapts objects at run-time of the application, does not teach or suggest "wherein the classes of the core version of the software indicated as being statically extensible are processed together with their corresponding extensions, and wherein the classes of the core version of the software indicated as being

dynamically extensible are processed separate from their corresponding extensions" as recited by claim 11. The adaptation manager describes adapting objects at run-time of the application, which does not teach or suggest "wherein the classes of the core version of the software indicated as being dynamically extensible are processed separate from their corresponding extensions" as recited by claim 11. In fact, Raverdy never mentions separate processing for classes that are dynamically extensible. Instead, Raverdy's "adaptive methods" are compiled into the executable, as Raverdy describes in Fig. 5 and the related description.

For at least these reasons, Raverdy, does not teach or suggest all limitations of claim 11. Therefore, claim 11 should be allowable.

Claim 23

Claim 23 has been amended with language from dependent claim 27. Amended claim 23 reads as follows (emphasis added):

> an object description language pre-processor operable for receiving extensions to classes of the core version of the software in an object description language and generating a source code version of the extensions to classes of the core version of the software, wherein the classes of the core version of the software are indicated in the object description language as being dynamically extensible at runtime or statically extensible prior to compile time, and *wherein the object description language pre-processor is programmed for processing the classes of the core version of the software indicated as being dynamically extensible separate from their corresponding extensions*.

As discussed above with regard to claim 11, Raverdy's description of an "adaptation manager" for adapting objects at run-time of the application does not teach or suggest "wherein the object description language pre-processor is programmed for processing the classes of the core version of the software indicated as being dynamically extensible separate from their corresponding extensions" as recited by claim 23. Therefore, claim 23 should be allowable.

Claim 35

Claim 35 has been amended primarily with language from dependent claims 37 and 39. Amended claim 35 reads as follows (emphasis added):

> wherein the extension declarations correspond to a particular configuration of the extended version of the software and the extension declaration further

> comprise one or more attribute declarations for indicating the configuration of the extended version of the software, and *wherein the core version of the software is an extensible core software development tool framework and the extended version of software is a customized software development tool and the configuration of the extended version of software comprises choosing a target processor architecture*.

For example, the Application page 5, line 24 to page 6, line 14, describes extending a core software framework for different configurations, including different target processor architectures, as follows:

> FIG. 1 illustrates an exemplary core software framework that may be extended to build custom compilers and other software development tools of multiple different configurations to reflect multiple software development scenarios. The core 110 provides an extensible architecture that can be used as a building block to build customized software development tools 111-114. The core 110 software can be extended by adding software extensions related to one or more software development scenarios. For example, a JIT (Just-In-Time) compiler 111 targeting an IA-64 processor 121 may be built by providing software extensions to the core 110. In this case, the fact that the compiler is a JIT compiler 111 and that is being targeted for a particular processor architecture (IA-64 processor at 121) may determine the form and content of the software extensions to the core 110. Thus, software extensions related to the JIT compiler scenario and the IA-64 target scenario may be used to specify a configuration for building a customized software development tool. Other scenario factors such as the source languages 101-104 and the features of the tool that may be turned on or turned off based on particular software development scenarios may also add complexity to the task of building custom software development tools by extending a standard core framework such as the one in FIG. 1.

Additional description of various target processor architectures can be found, for example, in the Application at page 6, lines 17-21.

Raverdy does not teach or suggest "wherein the extension declarations correspond to a particular configuration of the extended version of the software and the extension declaration further comprise one or more attribute declarations for indicating the configuration of the extended version of the software, and *wherein the core version of the software is an extensible core software development tool framework and the extended version of software is a customized software development tool and the configuration of the extended version of software comprises choosing a target processor architecture*" as recited by claim 35. Regarding the language from former dependent claim 39, the Examiner argues that Raverdy's Figure 5, element 530 (and

related text) describes this language. Action, page 11. Applicants respectfully disagree. The section of Raverdy cited by the Examiner describes "the process of compiling a DART application using the DART compiler." Raverdy, col. 9, lines 32-50. Regarding element 530, Raverdy describes, "adaptive service implementation files 530, which contain a set of pre-defined rules for processing the DART keywords." Raverdy, col. 9, lines 37-40. This section of Raverdy does not teach or suggest "wherein the core version of the software is an extensible core software development tool framework and the extended version of software is a customized software development tool and the configuration of the extended version of software comprises choosing a target processor architecture" as recited by claim 35. In fact, Raverdy never describes choosing a target processor architecture.

For at least these reasons, Raverdy, does not teach or suggest all limitations of claim 35. Therefore, claim 35 should be allowable.

Claim 41

Claim 41 has been amended with language from dependent claim 43. Amended claim 41 reads as follows (emphasis added):

> A computer readable storage medium having stored thereon software code for a pre-processor program, wherein the pre-processor program is operable for receiving, in an object description language, classes of a core version of a software and corresponding extensions to the classes of the core version of the software to be used for extending the core version of the software, wherein the classes of the core version of the software are indicated in the object description language as being statically extensible prior to compile time or dynamically extensible at runtime, and *wherein the pre-processor program is further operable for using the extensions of the classes of the core version of the software received in form of the object description language to generate a source code version of the extensions to be used for linking the extensions to their corresponding classes of the core version of the software at runtime to extend the core version of the software*.

Raverdy does not teach or suggest "*wherein the pre-processor program is further operable for using the extensions of the classes of the core version of the software received in form of the object description language to generate a source code version of the extensions to be used for linking the extensions to their corresponding classes of the core version of the software at runtime to extend the core version of the software*" as recited by claim 41. Regarding the

language from former dependent claim 43, the Examiner argues that Raverdy's Figure 5, element 570 (and related text) describes this language. Action, page 12. Applicants respectfully disagree. The section of Raverdy cited by the Examiner describes "the process of compiling a DART application using the DART compiler." Raverdy, col. 9, lines 32-50. Regarding element 570, Raverdy describes that the application is compiled to produce "an executable file." Raverdy, col. 9, lines 40-50. However, Raverdy's description of generating "an executable file" does not teach or suggest generating "a source code version of the extensions to be used for linking the extensions to their corresponding classes of the core version of the software at runtime to extend the core version of the software." In fact, Raverdy never mentions generating a source code version of extensions to use for linking at run-time. Instead, Raverdy's "adaptive methods" are compiled into the executable, as Raverdy describes in Fig. 5 and the related description.

For at least these reasons, Raverdy, does not teach or suggest all limitations of claim 41. Therefore, claim 41 should be allowable.

Claim 44

Claim 44 has been amended with language from claims 12 and 14. Amended claim 44 reads as follows (emphasis added):

> means for receiving extensions to classes of the core version of the software in an object description language and generating a source code version of the extensions to classes of the core version of the software, wherein the classes of the core version of the software are indicated in the object description language as being dynamically extensible at runtime or statically extensible prior to compile time, *wherein the classes of the core version of the software indicated as being statically extensible are processed together with their corresponding extensions, and wherein the classes of the core version of the software indicated as being dynamically extensible are processed separate from their corresponding extensions*;

As described above with regard to claim 11, Raverdy's description of an adaptive software platform for general purpose distributed applications does not teach or suggest "*wherein the classes of the core version of the software indicated as being statically extensible are processed together with their corresponding extensions, and wherein the classes of the core version of the software indicated as being dynamically extensible are processed separate from their corresponding extensions*" as recited by claim 44. Therefore, claim 44 should be allowable.

Dependent claims 2, 4-10, 13, 15-22, 24-26, 28-34, 36, 38, 40, and 42

Claims 2 and 4-10 ultimately depend on claim 1. Thus, for at least the reasons set forth above with regard to claim 1, claims 2 and 4-10 should be in condition 4-10.

Claims 13 and 15-22 ultimately depend on claim 11. Thus, for at least the reasons set forth above with regard to claim 11, claims 13 and 15-22 should be in condition for allowance. Applicant will not belabor the merits of the separate patentability of claims 13 and 15-22.

Claims 24-26 and 28-34 ultimately depend on claim 23. Thus, for at least the reasons set forth above with regard to claim 23, claims 24-26 and 28-34 should be in condition for allowance. Applicant will not belabor the merits of the separate patentability of claims 24-26 and 28-34.

Claims 36, 38, and 40 depend on claim 35. Thus, for at least the reasons set forth above with regard to claim 35, claims 36, 38, and 40 should be in condition for allowance. Applicant will not belabor the merits of the separate patentability of claims 36, 38, and 40.

Claim 42 depends on claim 41. Thus, for at least the reasons set forth above with regard to claim 41, claim 42 should be in condition for allowance. Applicant will not belabor the merits of the separate patentability of claim 42.

**Request for Interview**

If any issues remain, the Examiner is formally requested to contact the undersigned attorney prior to issuance of the next Office action in order to arrange a telephonic interview. It is believed that a brief discussion of the merits of the present application may expedite prosecution. Applicants submit the foregoing formal Amendment so that the Examiner may fully evaluate Applicants' position, thereby enabling the interview to be more focused.

This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

## Conclusion

The claims should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Cory A. Jones/
Cory A. Jones
Registration No. 55,307